# Speculation Techniques for Improving Load Related Instruction Scheduling

IDC Architecture Research

Adi Yoaz, Mattan Erez, Ronny Ronen, and Stephan Jourdan

ISCA-26 May 2, 1999

# Concept

## Improve the Scheduling Algorithm

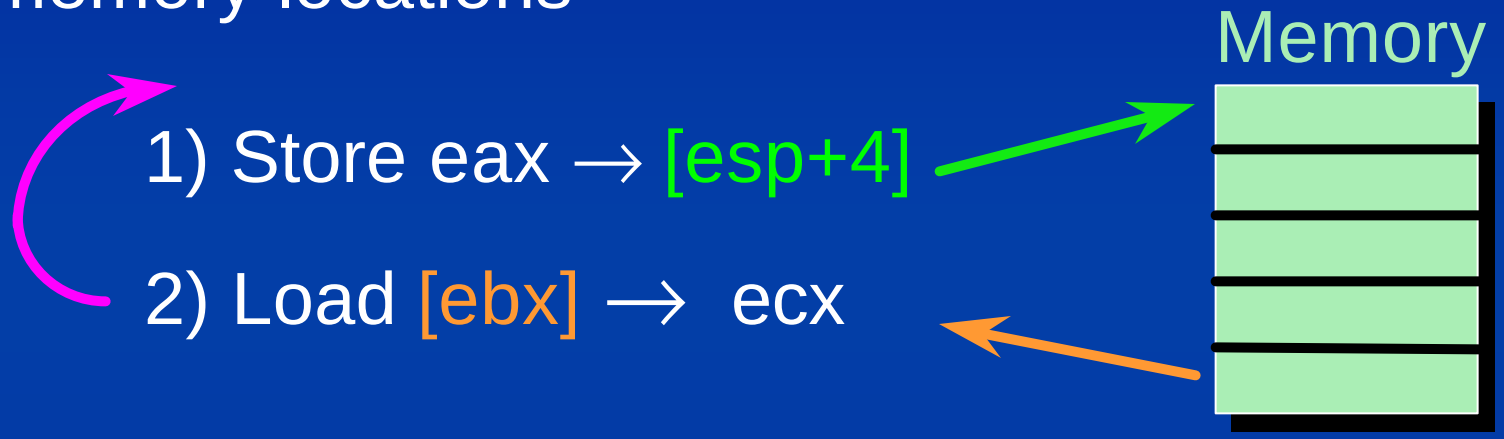
Provide new information to the scheduler, which is not available at schedule time in current implementations

# Agenda

- Speculative Memory Disambiguation
  - Simpler implementation of an existing concept
- Data Cache Hit-Miss Prediction
  - New concept
- Bank Prediction
  - New concept

# Memory Disambiguation

- Motivation: Execute a load instruction before preceding stores when they address different memory locations




- When [ebx] != [esp+4] performance may be improved by advancing the load before the store
- Problem: *Memory Ambiguity* - lack of information regarding memory dependencies at schedule time

We offer a cost effective solution

# P6 Family LD/ST Ordering Scheme

- Improves the conservative ordering model
- Store instructions depend both on source data and source address operands (store eax → [esp+4])
- A store-address can often be calculated before the data to be stored is ready
- A store instruction is split into two *UOPs*:
  - **STA (Store Address) & STD (Store Data)**


loads cannot bypass STAs

1 STA [esp+4]
2 STD eax
3 Load [ebx] → ecx


loads can bypass STDs

1 STA [esp+4]
2 STD eax
3 Load [ebx]→ ecx

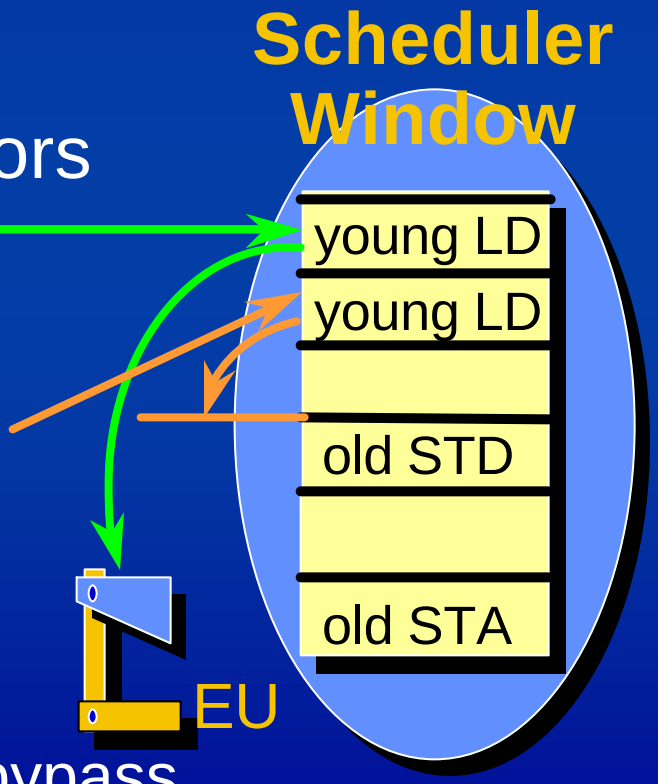
Problem: The load and the store may reference the same memory address introducing a hazard

# Idea: Simple Dependency Predictor

- Previous studies concentrated on load-store pairing
- Our idea: predict load-store *Collisions*
  - Based on collision history

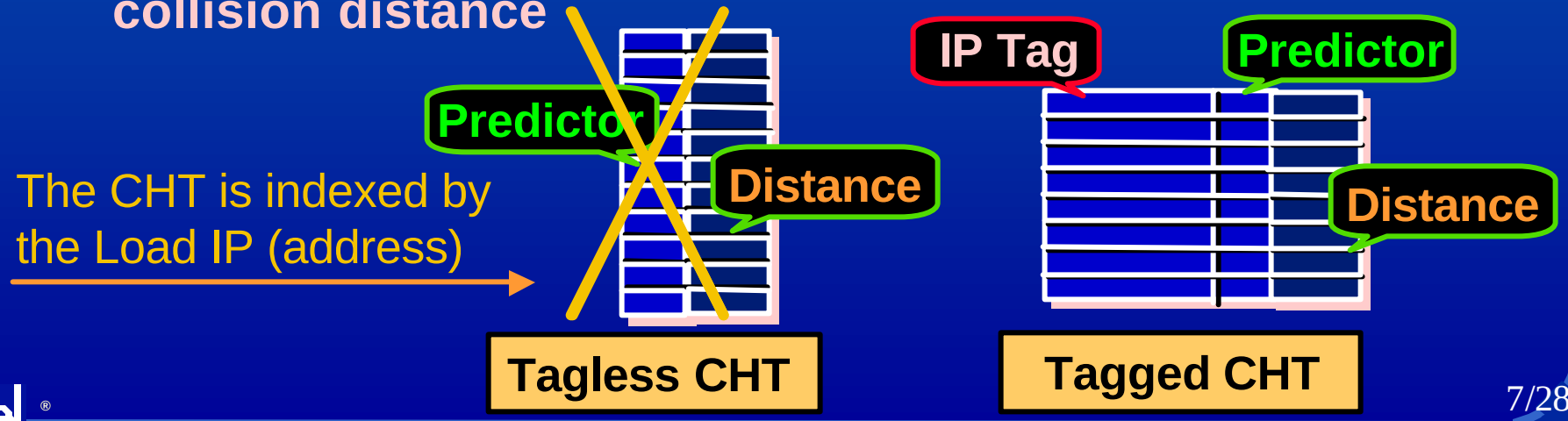
**Cost Effective Solutions:**

- Option #1: Simple binary predictors
  - Advance loads predicted as non-colliding
  - Delay loads predicted as colliding
- Option #2: Enhanced predictors
  - Tells how many stores a load can bypass
  - May perform efficient load-store pairing



# Memory Dependency Predictor

- Collision History Table: record load collision histories
  - Each CHT entry contains a predictor (saturating counter)
  - **Can be combined with the load op-code in the cache**
- Simple CHT - **inclusive** collision predictor
  - Provides a **"general"** yes/no prediction
  - "Predicted Colliding" loads cannot bypass any older store
- Enhanced CHT- **exclusive** collision predictor
  - Provides a **"specific"** ***collision distance*** prediction
  - "Predicted Colliding" loads can bypass older stores within this **collision distance**



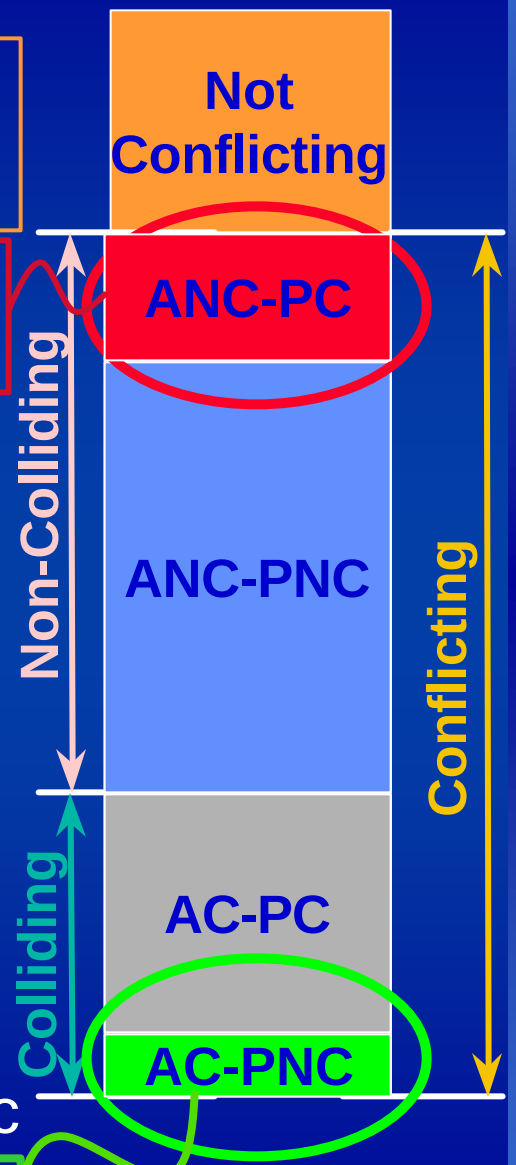
# Terminology

- Load classification:
  - **Not Conflicting**
  - **Conflicting**
    - Actually Non-Colliding
      - **Predicted Colliding (ANC-PC)**
      - **Predicted Non Colliding (ANC-PNC)**
    - Actually Colliding
      - **Predicted Colliding (AC-PC)**
      - **Predicted Non-Colliding (AC-PNC)**
- Goal: minimize wrong predictions
  - The misprediction penalty is not symmetric

No prior stores in the dynamic scheduling window

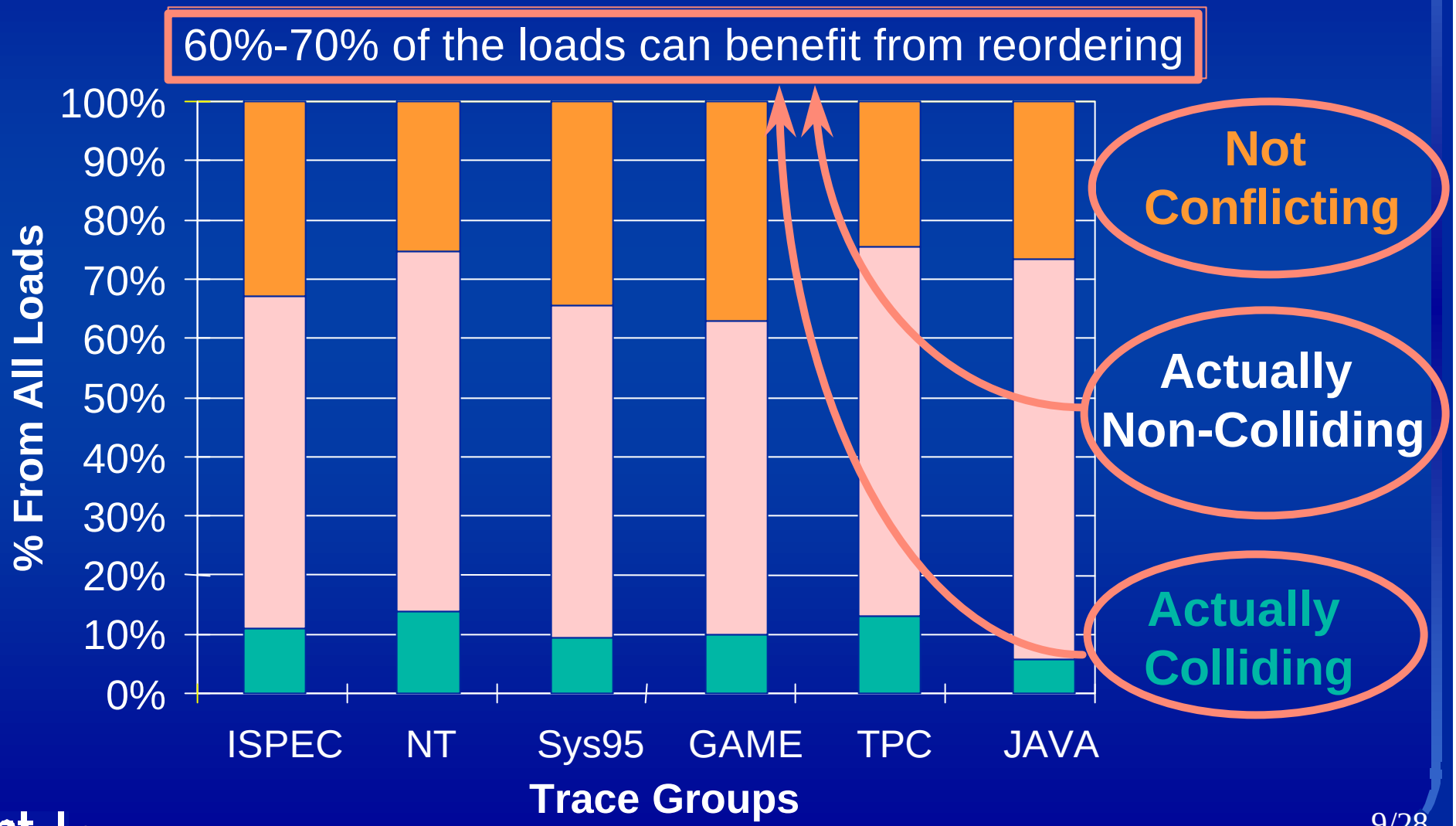
Lost opportunities to increase parallelism

Re-Execution penalty

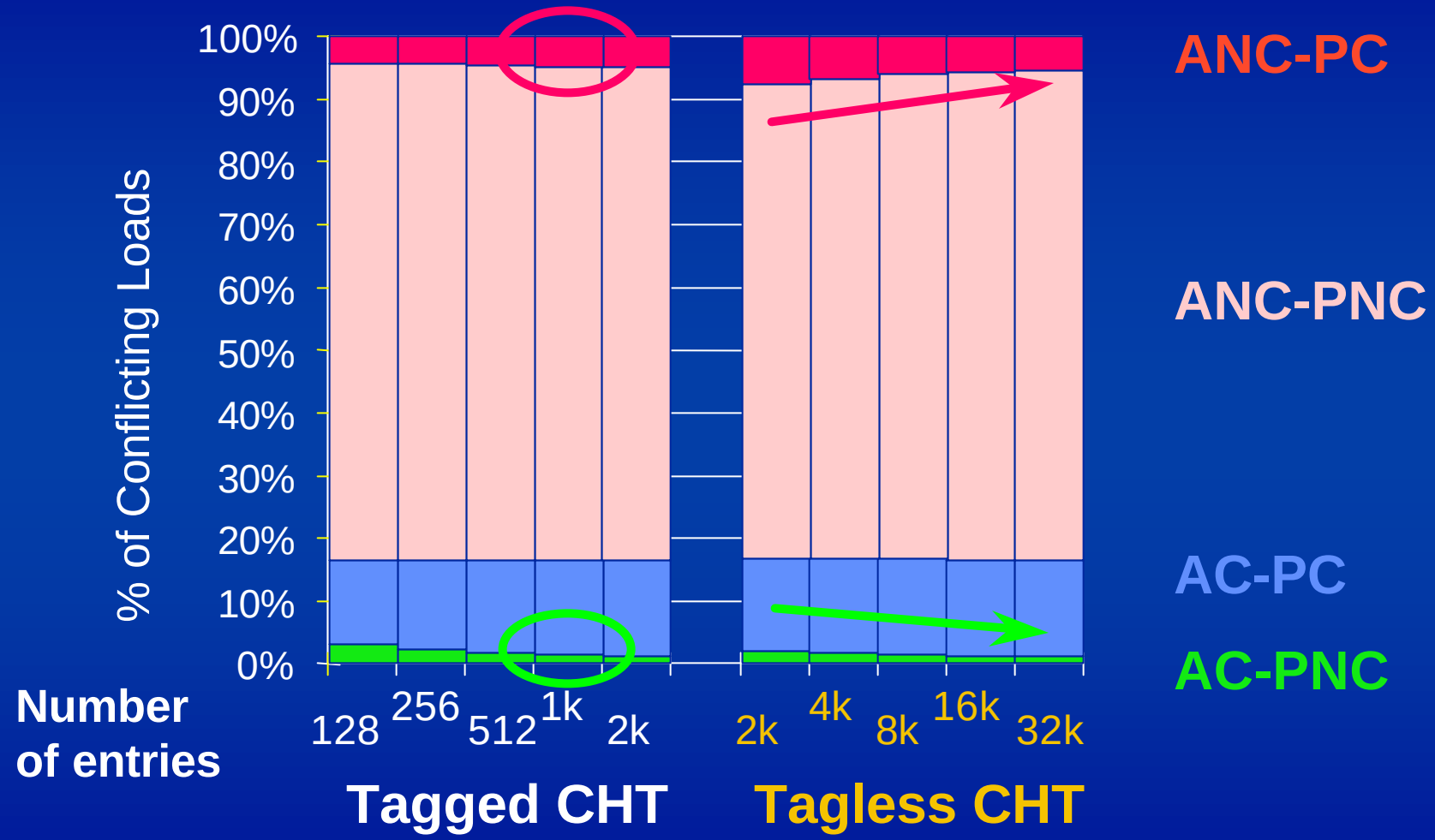


# Potential Area For Improvement

**Machine model: 32 entry scheduling window, 2 int, 2 mem EU's**



# Simple CHT Prediction Accuracy

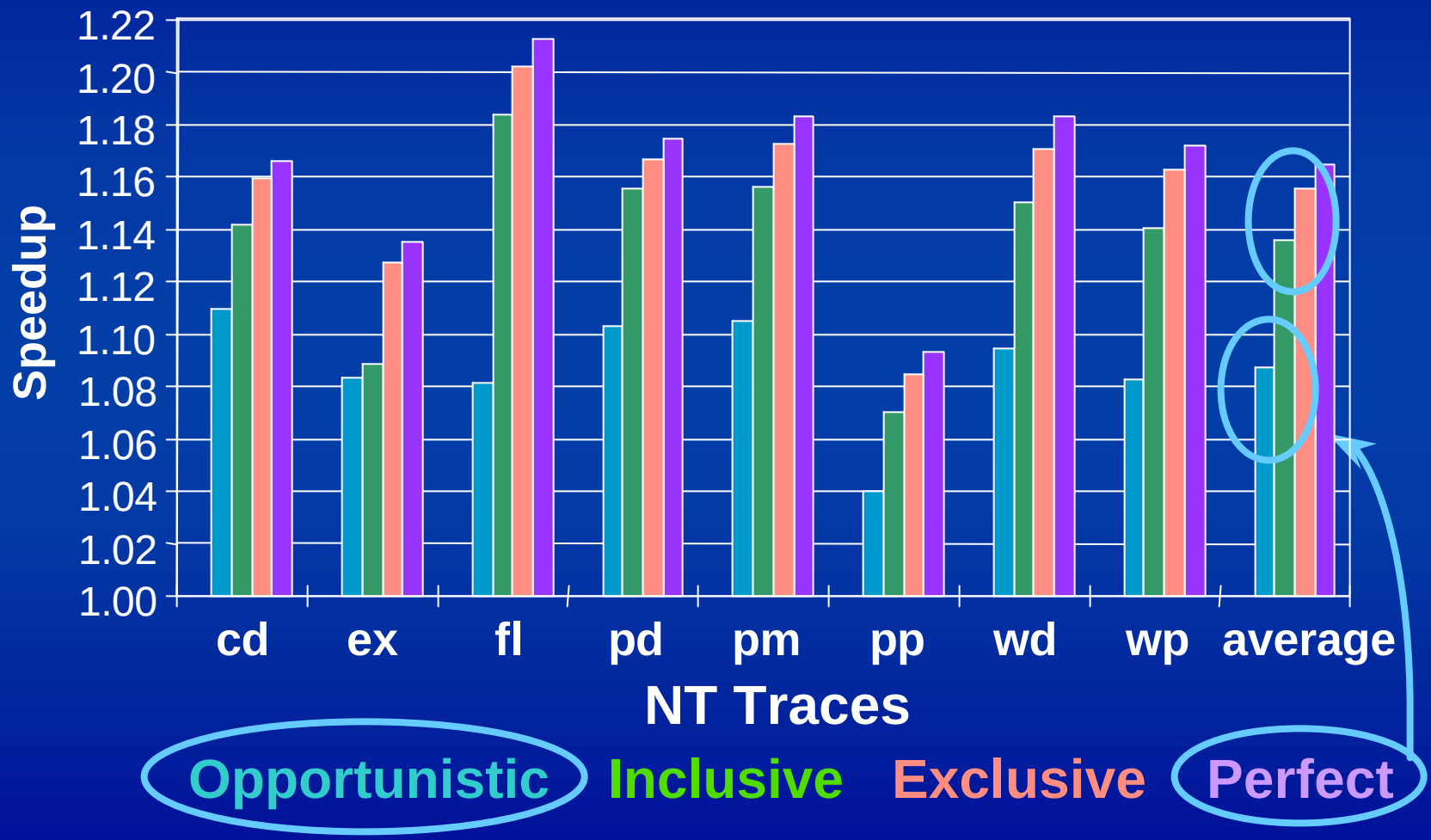


# Simulation Environment

- Simulation Tool for uArch Research
  - General Out-Of-Order machine
  - Detailed memory hierarchy model
  - Advanced branch prediction model
- Basic machine configuration
  - Scheduler: 32 entries window size (“RS” size)
  - Execution: 2 integer, 2 memory units
  - Renamer Register Pool: 128 entries (“ROB” size)
  - Memory: 16K Icache, 16K Dcache and 256K unified L2
  - Front end: 4 uops fetched and renamed per clock
  - Retire Rate: up to 4 uops per clock
  - Collision predictor: 1K entry Tagged CHT

# Performance Results

**Baseline performance is for the P6-like scheme**



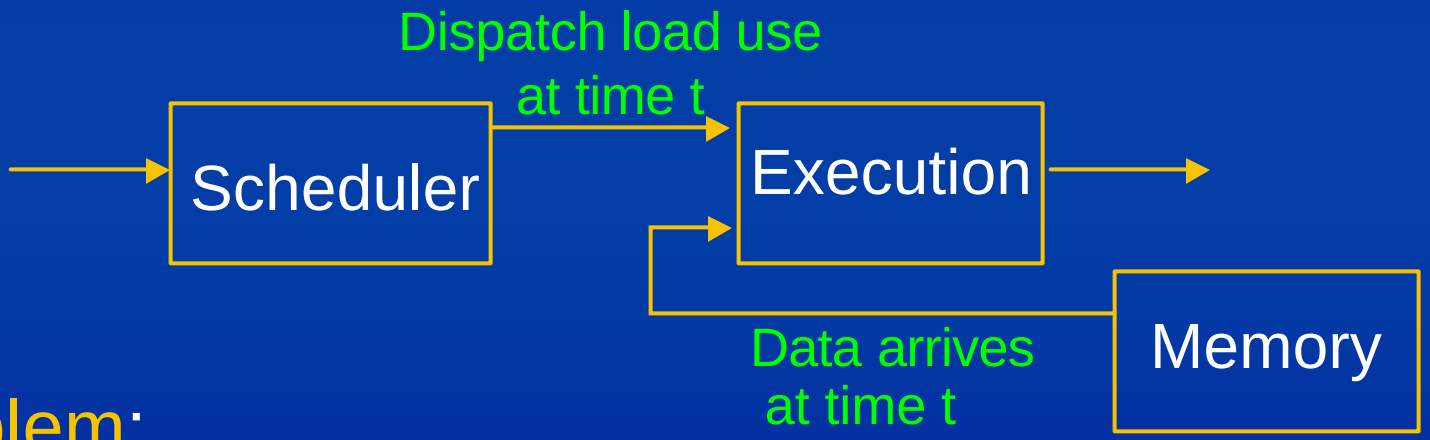

**Collision predictor: 1K entry Tagged CHT**

# Disambiguation - Summary

- Provides a large performance increase (~17%)
- Memory dependencies are highly predictable
- Simple (implementable) methods perform well:
  - **Inclusive scheme - 14% speedup (82% of potential)**
    - **The simplest configuration uses only 1-bit per load**
    - **Can be integrated with the load op-code in the cache**
  - **Exclusive scheme - 16% speedup (95% of potential)**
    - **Predicts "how far" a load can be advanced**
- Asymmetric misprediction penalty
  - Some predictors optimize the number of collisions caught
  - Others maximize the opportunities to increase parallelism

# Hit-Miss Prediction

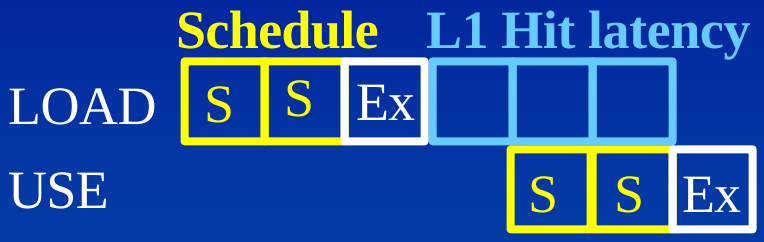
- Motivation: Schedule load dependent instructions to execute exactly when the data is retrieved
  - Improve performance through better scheduling
  - Achieve better resource utilization



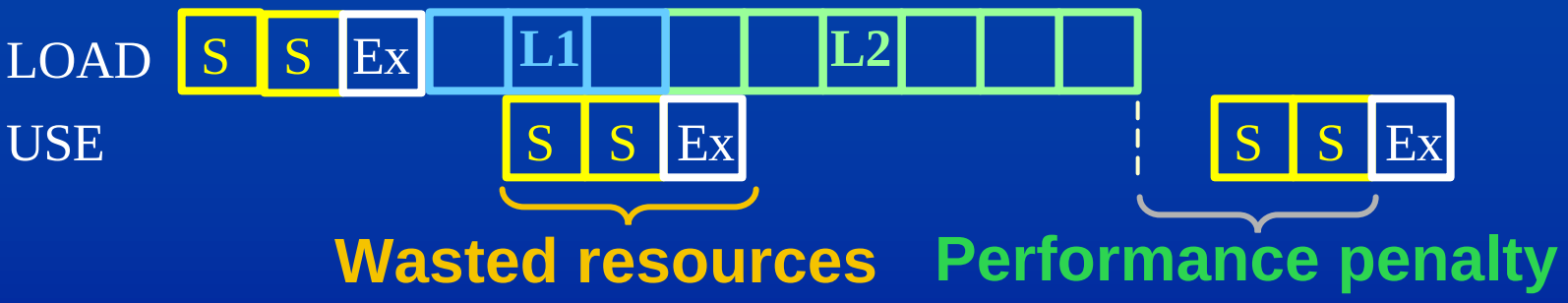

- Problem:
  - Efficient scheduling relies on knowing instruction latencies
  - Load operations have dynamic latencies
    - Sub-optimal scheduling
    - Poor resource utilization

# Current Scheduling method

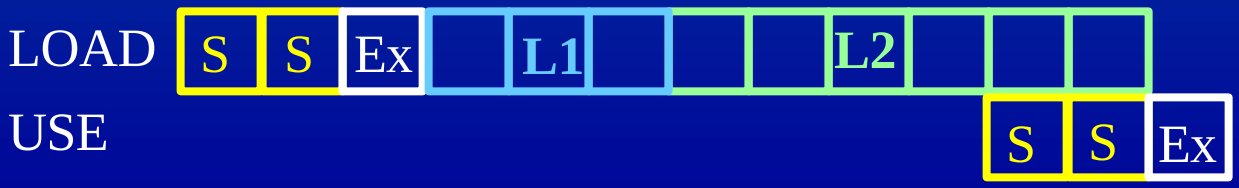
- A hit latency is assumed for all loads




- On a cache miss re-scheduling and re-execution of the load dependent instructions is required




- Efficient scheduling of the use inst. on a cache miss



# Idea: Hit-Miss Prediction

- Predict dynamic load latencies
- Use a Hit Miss Predictor (per-load binary prediction)
  - if a load instruction is predicted as a cache miss, its dependent instructions are delayed
- Allows efficient scheduling of load dependent instructions based on the associated load latencies
- Reduces re-executions and re-scheduling
  - increases the effective execution bandwidth
  - saves many lost cycles caused by re-scheduling
  - lowers power consumption

# Load Classification

Based on prediction and actual outcome

- AH-PH: Actual Hit - Predicted Hit
  - Same performance impact as current implementation
- AM-PH: Actual Miss - Predicted Hit - uncaught misses
  - Same performance impact as current implementation =>
  - Lost  opportunities to improve performance
- AM-PM: Actual Miss - Predicted Miss - misses caught by the predictor
  - Performance boost
- AH-PM: Actual Hit - Predicted Miss - load dependent instructions are dispatched later than they could have
  - Slight performance degradation
  - No recovery required

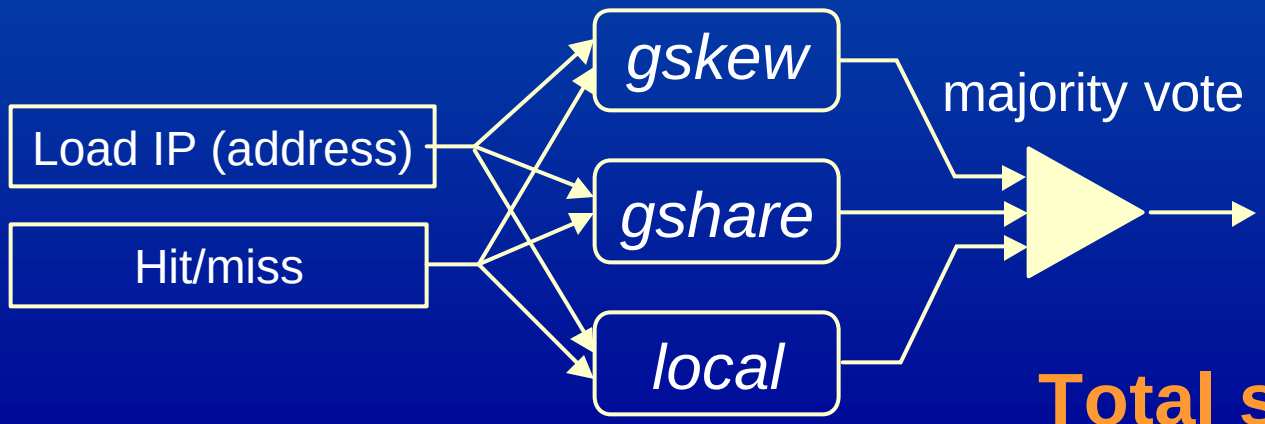
# Hit-Miss Predictor Example (local)

- Two level local history (as in branch prediction)
  - First level is an untagged local history table
  - Second level is a shared prediction table
- A simple and cheap predictor
  - Tested a ~2KB predictor (2048 entries, 8 bit history)
- Provides relatively accurate predictions


**Two level local predictor**

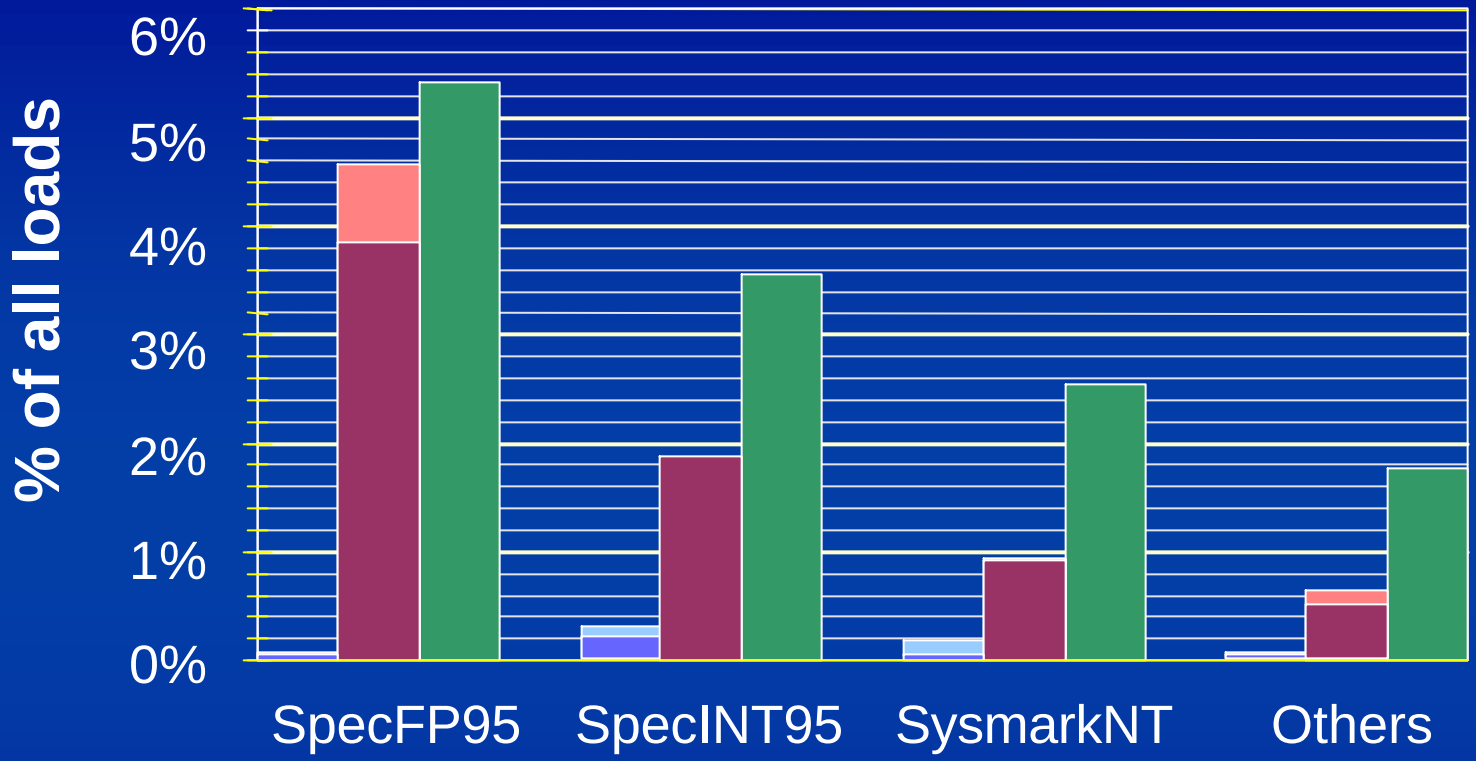
# Hybrid Example

- The hybrid improves prediction accuracy
- Based on a majority vote between three predictors
  - local predictor (Yeh & Patt)
    512 entries, 8 bit history (~0.5KB)
  - gshare predictor (McFarling)
    11 bit history (0.5 KB)
  - gskew predictor (Michaud & Seznec)
    three 1K entry tables and a 20 bit history (0.75KB)




**Total size ~1.75KB**

# Hit-Miss Prediction Accuracy



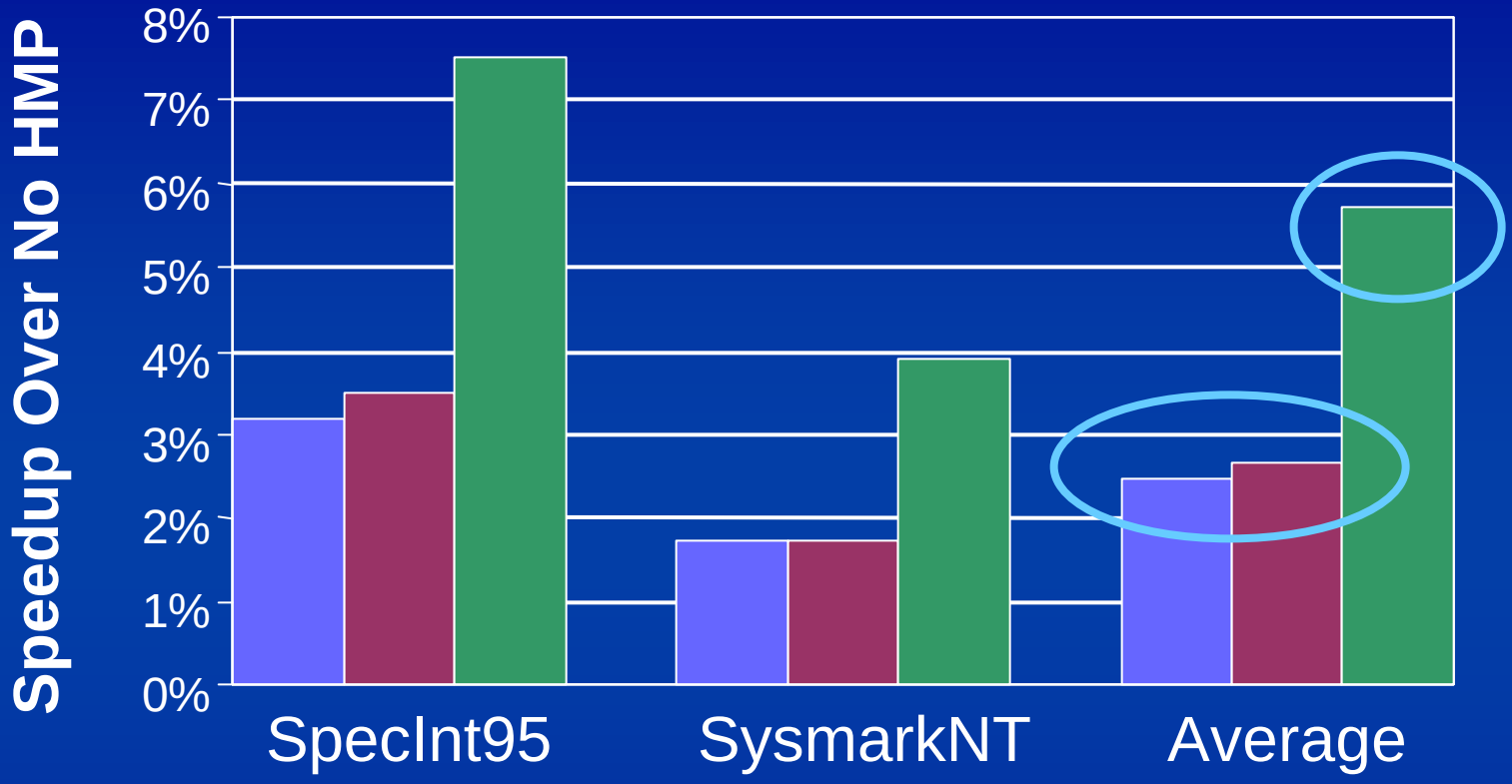

| | AH-PM | AM-PM | Misses |
|---|---|---|---|
| Local | | | |
| Hybrid | | | |

Both predictors are about 2KB

The local catches 34%-85% of the misses and mispredicts 0.3%-0.07% of the hits

The hybrid significantly reduces the number of mispredicted hits (0.2%-0.04%)

# Hit-Miss Prediction - Speedup




Both predictors are about 2KB

**Base line performance is for a machine with: perfect memory disambiguation and 8-cycles penalty for re-execution recovery**

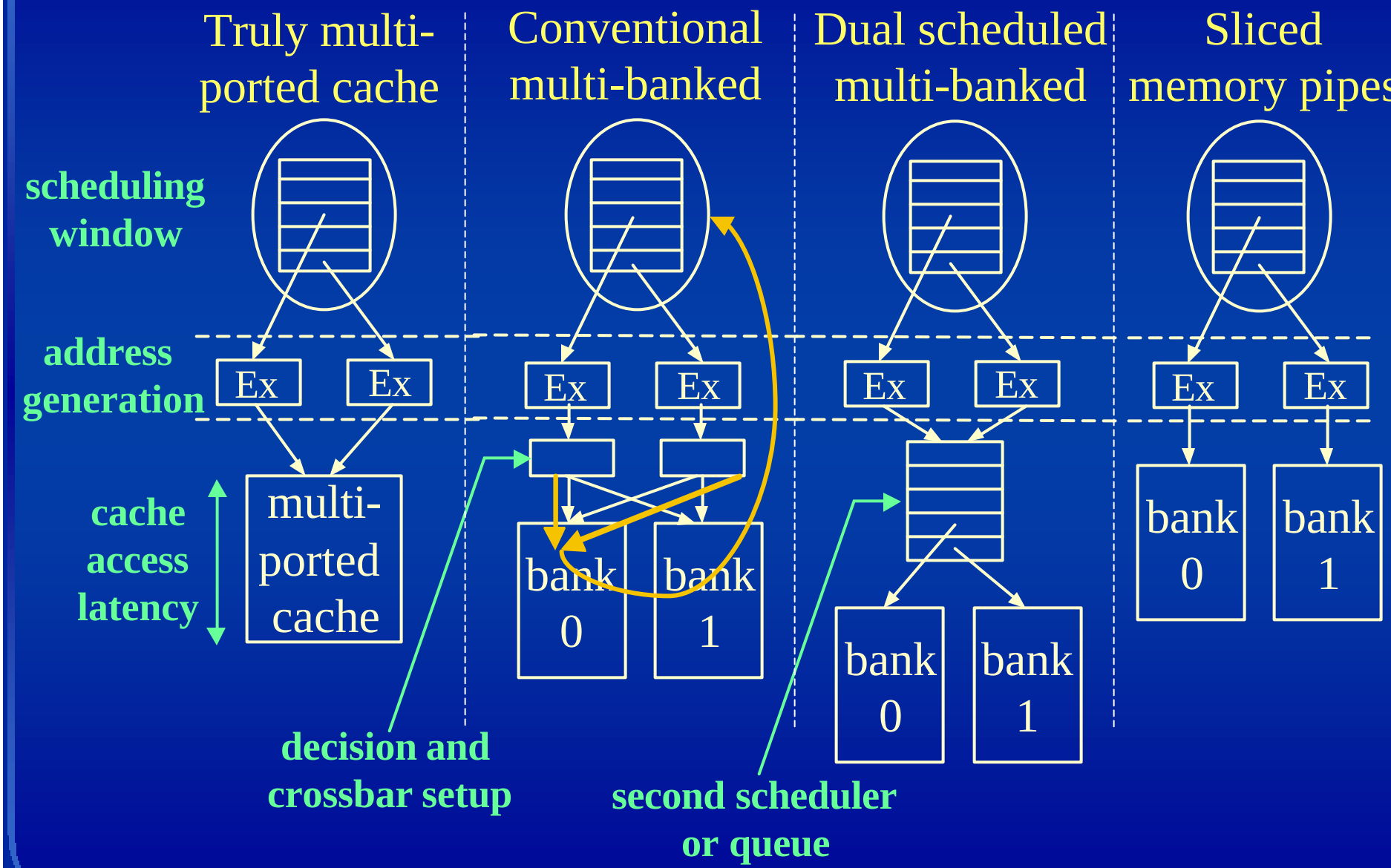
# HMP - Summary

- Hit Miss Prediction improves performance through
  - Better scheduling of load dependent instructions
  - A reduction in the required execution bandwidth
- A simple local predictor exhibits good results in terms of number of misses caught
- A hybrid predictor reduces the number of AH-PM loads significantly
- Perfect prediction has large performance potential
- Simple predictors yield moderate speedup

# Cache-Bank Prediction

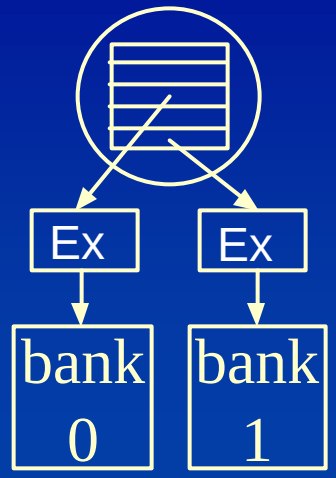
- Motivation: Allow a multi-banked cache to approach the performance of a truly multi-ported cache
- Problem:
  - Multi-ported caches are expensive and hard to scale
  - Conventional multi-banked caches
    - Require a crossbar stage or a second scheduler that increase the memory access latency
    - Suffer from bank conflicts that reduce performance
- Bank Prediction:
  - Increases the efficiency of multi-banked caches by scheduling memory accesses to independent banks
  - Provides a way for simplifying and scaling the memory execution pipelines

# Multi-Bank Design Options



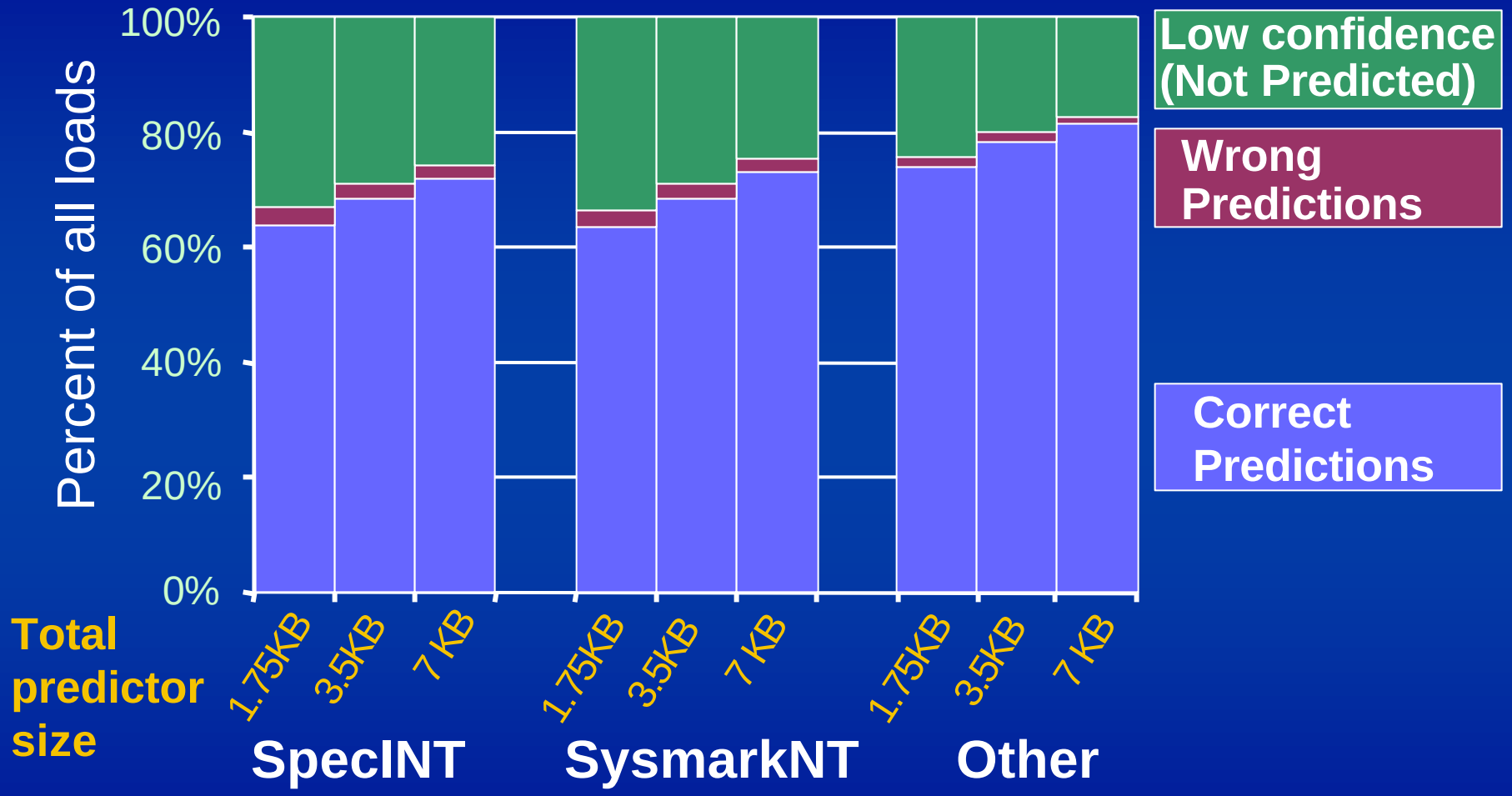
# Bank Prediction effects

- Efficient scheduling for better utilization of the multiple banked structure
- Simplified memory pipeline through the *slicing* of the memory HW structures
  - Reduced memory related latencies
    - No need for a cross-bar or a second scheduler
    - Smaller CAMs in the disambiguation and memory ordering HW
  - Highly scalable scheme




- On mispredictions, loads should be re-scheduled
  - Minimize re-executions by duplicating *low-confidence loads* to all memory pipelines

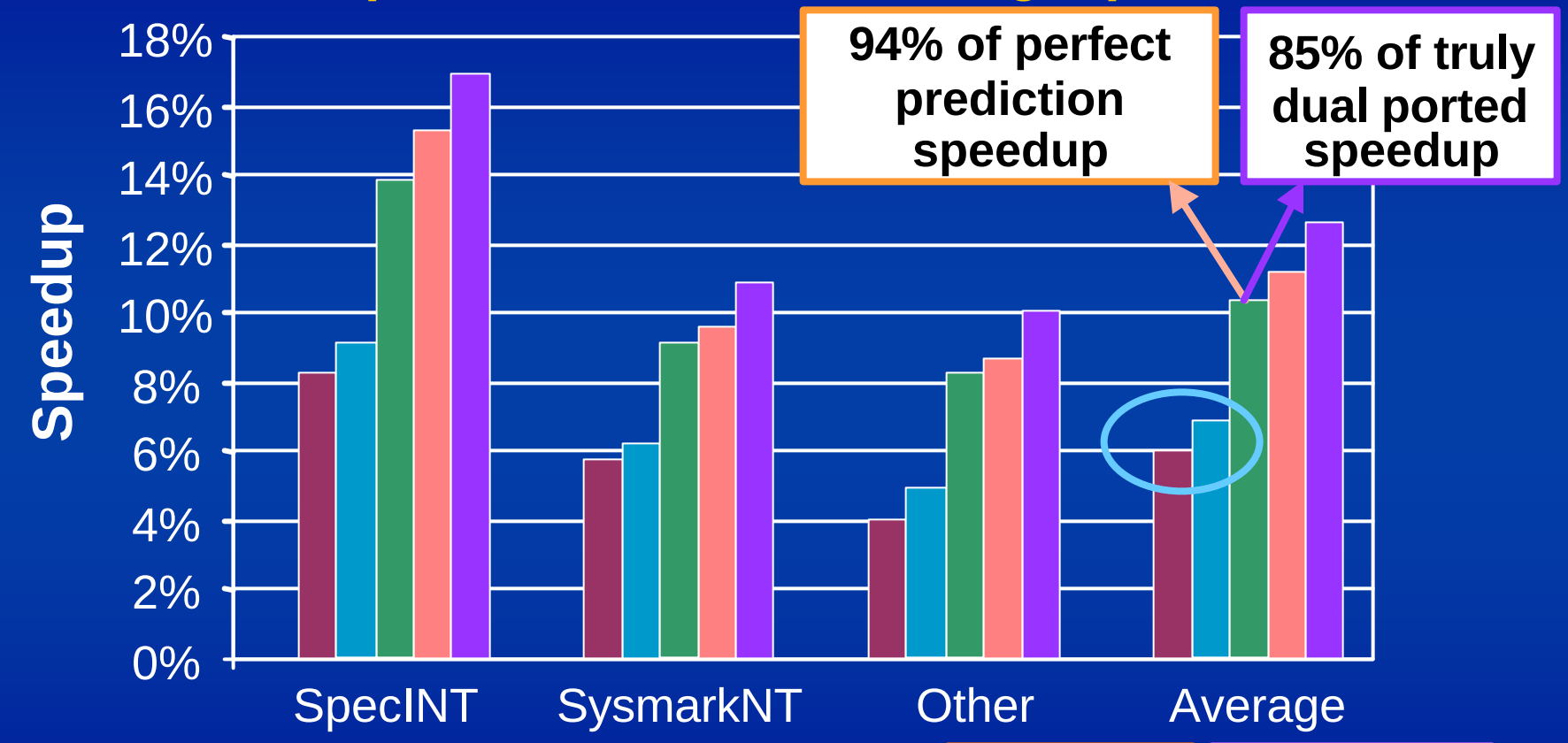
# Bank Predictor Results




The bank prediction is made using a hybrid comprised of a: ***local***, ***gshare***, and ***gskew*** predictors.

# Bank Prediction - Speedup

**Base line performance is for a single ported cache**




| Dual-banked w/ a crossbar | Dual-banked w/ dual scheduler | Sliced Memory | Perfect Prediction | Ideal: Truly dual ported |
|---|---|---|---|---|

**Tested 2KB Hybrid Predictor**

# Bank Prediction - Summary

- Bank Prediction improves performance through
  - Efficient scheduling of loads to independent cache banks
  - Allows the simplification of memory pipeline
- A simple predictor shows good results in terms of accuracy and performance speedup
  - The hybrid predictor is less than 2KB in size
  - Closely approaches the truly multi-ported cache performance